# **Product Data Explorer — Full-Stack Assignment**

## **Goal**

Build a production-minded product exploration platform that lets users navigate from high-level headings → categories → products → product detail pages powered by **live, on-demand scraping**.

All scraping **must be done against** [**World of Books**](https://www.worldofbooks.com/).

* You need to use the technologies mentioned in this document.

Candidates must submit:

* **GitHub repo link** (public or private with access)
* **Deployed project link** (live frontend + working backend)
* **Both links should be submitted through the following Google Form:** <https://forms.gle/ndGcbU1kgViQBK939> .

## **High-level requirements (must have)**

### **Frontend**

* Tech: **React (Next.js, App Router), TypeScript, Tailwind CSS**.
* Core pages/components:  
  + Landing / Home (shows navigation headings)
  + Category drilldown pages
  + Product grid / results (supports paging / limit)
  + Product detail page (reviews, ratings, recommendations, metadata)
  + About / Contact / README page in site
* UX:  
  + Responsive (desktop & mobile), accessible (WCAG AA basics), skeleton/loading states and smooth transitions.
  + Persist user navigation & browsing history client-side and via backend for reloading.
  + Use a client data fetching strategy (SWR or React Query recommended).
* Deliverables:  
  + Live URL to the deployed frontend
  + Instructions in README: how to run locally, environment variables, build steps

### **Backend**

* Tech: **NestJS (Node + TypeScript)**.
* Production-ready DB: PostgreSQL, MongoDB, or other (must justify choice).
* Expose RESTful endpoints (see example API contract below).
* On relevant calls, trigger a **real-time scrape** (Crawlee + Playwright) and store results. Allow:  
  + On-demand scrapes triggered by user actions
  + Safe caching to avoid excessive scraping
* Robust engineering:  
  + Proper DTO validation, error handling, logging, and resource cleanup
  + Concurrency handling, deduplication of scrape results, idempotency where applicable
  + Rate limiting / backoff for external sites and queueing of long running scrapes

### **Scraping (World of Books)**

* **Target site:**<https://www.worldofbooks.com/>
* Use **Crawlee + Playwright** (or equivalent headless browser framework).
* Extract and persist:  
  + Navigation headings (e.g., “Books”, “Categories”, “Children’s Books”, etc.)
  + Categories & subcategories
  + Product tiles/cards:  
    - Title, Author, Price, Image, Product Link, Source ID
  + Product detail pages:  
    - Full description
    - User reviews & ratings (if present)
    - Related/recommended products
    - Any additional available metadata (publisher, publication date, ISBN, etc.)
* Save all scraped data into your DB with relationships & unique constraints.
* Implement **deduplication** and **caching with expiry** so repeated scrapes don’t overload World of Books.
* Provide a way to re-fetch updated product data on demand.

⚠️ **Ethical scraping reminder**:

* Respect robots.txt and terms of service.
* Use proper rate limiting and delays.
* Implement retries and exponential backoff.
* Cache results wherever possible to avoid hitting the site repeatedly.

## **Suggested Database schema (entities)**

* navigation — id, title, slug, last\_scraped\_at
* category — id, navigation\_id, parent\_id, title, slug, product\_count, last\_scraped\_at
* product — id, source\_id, title, price, currency, image\_url, source\_url, last\_scraped\_at
* product\_detail — product\_id (FK), description, specs (json), ratings\_avg, reviews\_count
* review — id, product\_id, author, rating, text, created\_at
* scrape\_job — id, target\_url, target\_type, status, started\_at, finished\_at, error\_log
* view\_history — id, user\_id (optional), session\_id, path\_json, created\_at
* Add indexes on source\_id, last\_scraped\_at, and unique constraints on source\_url/source\_id.

## **Non-functional requirements**

* **Security:** sanitize inputs, secure environment variables, do not commit secrets, enable CORS properly, minimal rate limiting.
* **Performance & caching:** caching layer in DB (or Redis) with explicit expiry; avoid re-scraping unchanged pages.
* **Observability:** logging, basic metrics, and error tracking (console + file or a service).
* **Reliability:** queue/worker model for scrapes (do not block request thread); idempotent jobs.
* **Accessibility:** semantic HTML, keyboard nav, alt on images, color contrast.

## **Deliverables**

1. **GitHub repo link** with:  
   * frontend/ and backend/ folders
   * CI pipeline (GitHub Actions) for lint/test/build (recommended)
   * README with architecture overview, design decisions, and deployment instructions
   * Database schema and sample seed script
   * API documentation (Swagger or markdown)
   * Tests (unit + a couple integration tests)
   * Dockerfiles (bonus, but preferred)
2. **Deployed project link(s)**:  
   * Frontend URL (production)
   * Note: Must be live at submission time

## **Acceptance checklist (must pass)**

* Landing loads navigation headings (from World of Books via backend)
* Drilldown loads categories/subcategories (from World of Books via backend)
* Product grid displays real products (scraped from World of Books)
* Product detail page includes description, reviews/ratings, recommendations (scraped from World of Books)
* DB persists all scraped objects reliably
* On-demand scrape can refresh a product/category
* Frontend responsive and accessible baseline
* README + deploy links + API docs present
* Repo builds and runs with the provided instructions

## **Evaluation rubric (weights)**

* **Correctness & completeness (35%)** — feature coverage vs requirements
* **Architecture & engineering quality (20%)** — code structure, DTOs, validation, error handling
* **Scraping reliability & design (15%)** — safe scraping, queueing, dedupe, caching
* **UX & accessibility (10%)** — responsiveness, loading states, accessibility basics
* **Docs & deploy (10%)** — README, API docs, deployed links working
* **Tests & CI (10%)** — basic tests and CI pipeline

## **Bonus (highly valued)**

* Product search + rich filters (price range, rating, author)
* Intelligent caching / refresh strategy (DB-backed TTL, conditional scraping)
* SWR / React Query with optimistic UI updates
* Personalized recommendations or similarity engine (simple content-based OK)
* Full Docker setup for frontend, backend, DB + docker-compose
* Comprehensive test coverage (unit + e2e)
* API versioning and OpenAPI/Swagger with examples
* CI-based deploy to Vercel / Render / Heroku / Railway / Fly.io

## **Tips & constraints for candidates**

* **Be kind to World of Books**: implement delays, backoff, and cache results.
* **Focus on core features first** before optional bonuses.
* **Do not commit secrets** — use .env.example.
* **Include a fallback seed script** so reviewers can test even if scraping fails during review.